PROVIZ: AN INTEGRATED GRAPHICAL PROGRAMMING, VISUALIZATION AND SCRIPTING FRAMEWORK FOR WSNS

A Thesis
Presented to
The Academic Faculty

by

Ramalingam Kumbakonam Chandrasekar

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2013

PROVIZ: AN INTEGRATED GRAPHICAL PROGRAMMING, VISUALIZATION AND SCRIPTING FRAMEWORK FOR WSNS

Approved by:

Raheem A. Beyah,
Advisor and Committee Chair
School of Electrical and Computer Engineering
Georgia Institute of Technology

Edward J. Coyle
School of Electrical and Computer Engineering
Georgia Institute of Technology

Ying Zhang
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: 1 April 2013

This thesis is dedicated to my father, Chandrasekar, and my mother, Sadhana. You have been very patient and supportive on all the decisions that I have made till now, and I would like to thank you, for all the sacrifices that you have made for the sake of my welfare. Your unconditional love, constant motivation, and unwavering encouragement are what the reason for my success. Also, I dedicate this thesis to my younger brother, Arun. You have been a loving brother and a nice friend for me.

In loving memory of my grandparents, Chandrasekar and Mallika.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Raheem A. Beyah for his constant support and encouragement. Also, I would like to thank my mentor, Dr. Arif Selcuk Uluagac for his guidance and motivation. I am very much delighted that they offered me complete freedom to design and architect my work, and gave innovative and intriguing ideas to fine tune my research. Also, I would like to thank them for their advice and guidance, which made me excel as a good researcher.

I would like to extend my gratitude to Dr. Edward J. Coyle and Dr. Ying Zhang, for being on my MS Thesis Reading Committee. I would like to specially thank Dr. Edward J. Coyle, for his valuable comments towards the betterment of my research work.

I would like to extend my special thanks to Dr. Yang Wang and his group, for providing invaluable suggestions and inputs for my research, which motivated me to add more useful features to my work.

As a proud member of Communications Assurance and Performance (CAP) group, I would like to thank all my energetic colleagues for their advice and friendship, which provided a conducive atmosphere for me to proceed with my research.

I would like to thank all my family members and friends, for their understanding and motivation. Last but not least, I would like to thank the almighty for his blessings and kindness.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Wireless Sensor Networks (WSNs) are rapidly gaining popularity in various critical domains like health care, critical infrastructure, and climate monitoring, where application builders have diversified development needs. Independent of the functionalities provided by the WSN applications, many of the developers use visualization, simulation, and programming tools. However, these tools are designed as separate stand-alone applications, which force developers to use multiple tools. This situation often poses confusion and hampers an efficient development experience. To avoid the complexity of using multiple tools, a new, extensible, multi-platform, scalable, and open-source framework called PROVIZ is designed. PROVIZ is an integrated visualization and programming framework with the following features: PROVIZ 1) visualizes sensor nodes and WSN traffic by parsing the data received either from a packet sniffer (e.g., a sensor-based sniffer, or a commercial TI SmartRF 802.15.4 packet sniffer), or from a simulator (e.g., OMNeT); 2) visualizes a heterogeneous WSN consisting of different sensor nodes sending packets with different packet payload formats; and 3) provides a programming framework, which provides a graphical and script-based programming functionality, for developing WSN applications. Also, PROVIZ includes built-in extensible visual demo deployment capabilities that allow users to quickly craft network scenarios and share them with other users. Additionally, a secure and energy efficient wireless code dissemination protocol, named SIMAGE, was developed. SIMAGE is used by PROVIZ to wirelessly reprogram the sensor nodes. SIMAGE uses a link quality cognizant adaptive packet-sizing technique along with energy-efficient encryption protocols for secure and efficient code dissemination. In this thesis, the various features of PROVIZ's visualization and programming framework are explained, the functionality and performance of SIMAGE protocol is described, an example WSN security attack scenario is analyzed, and how PROVIZ can be used as a visual debugging tool to identify the security attack and aid in providing a software fix are discussed.

CHAPTER I

INTRODUCTION

Wireless Sensor Networks (WSNs) are used in various applications like remote health monitoring, volcanic activity monitoring, and critical infrastructure monitoring. Hence, the WSN researchers' community comprises of advanced WSN application developers who develop scalable and reliable WSN algorithms and applications, and developers who use WSNs for simple environment data gathering. In order to develop, test, visualize, and monitor a WSN application, a developer may need a programming tool, a simulator [1, 2], and a visualization tool [3, 4]. However, such tools are currently available as separate standalone tools, and the WSN researchers' have to use multiple tools for working with WSNs. Therefore, to aid in the process of WSN application development a new framework, which provides visualization, monitoring, and programming functionalities in a common platform, is required. Furthermore, as sensors are resource-constrained devices running with a limited source of power, critical applications can not afford the failure of any sensor, or sensor application. Thus, for visual debugging purposes a framework, which can visualize the WSN by continuously monitoring the network activity, is required. Also, after identifying a WSN application failure by visual debugging, the integrated framework should be capable of reprogramming the sensor nodes securely over-the-air. Wireless code dissemination protocols like those described in [5, 6] are available to reprogram the sensor nodes over-the-air without removing them from their deployment area. But, during the process of wireless code dissemination, these existing protocols do not consider the nodes with poor link quality. This oversight hinders the dissemination process because of the increased retransmissions required due to the poor link conditions. Also, the protocol described in [5] is not secure and can be eavesdropped by an attacker, and the protocol described in [6] provides secure code dissemination, but does not use energy-efficient encryption algorithms. To overcome these limitations, an energy-efficient and secure wireless code dissemination protocol, which

can work with the integrated framework to securely reprogram the sensor nodes is also required.

1.1 Research Objective

The objective of this thesis is to develop an integrated framework, which can: 1) visualize a heterogeneous WSN consisting of different sensors sending packets with different packet payload formats; 2) develop WSN applications using a graphical programing tool, or a script-based programing language; 3) simulate the WSN application using a simulator and visualize the simulation; and 4) program the sensor nodes over-the-air using wireless code dissemination protocol.

1.1.1 Summary of Research Contributions

The contribution of this research is two-fold. First, we develop an integrated programming, simulation, and visualization framework for WSNs. Second, we develop an energy-efficient, link quality cognizant, and secure wireless code dissemination protocol for programing the sensor nodes over-the-air.

1.1.1.1 PROVIZ: Integrated Framework

In this thesis, an integrated framework for WSNs, named PROVIZ [7], is developed. PROVIZ is novel in its approach by integrating the advantages of a graphical programming tool, visualization tool, and a script-based programming language into a common platform. PROVIZ is designed to work on top of the M-Core middle-ware [8], and hence, it reaps the advantage of using a service-based architecture. The graphical programming functionality can be used to graphically develop a WSN application, and the visualization functionality can monitor the WSN. Also, PROVIZ supports a scripting language, which can be used by advanced programmers for developing WSN applications.

PROVIZ has visualization functionality, which displays the sensor nodes in a WSN and the wireless packet transfers between the sensors by parsing the packet data from sniffers, or from the OMNeT [9] simulator. The sniffer can be either a live sensor-based sniffer, or the popular TI SmartRF 802.15.4 packet sniffer [10]. Since the visualization tool can be used

for live-monitoring of a WSN, it can also be used for visual debugging of various network error scenarios. Moreover, PROVIZ includes built-in extensible visual demo deployment capabilities, which allow users to quickly craft network scenarios. It is possible to export these demo scenarios as XML files so that users can easily modify and even share them among themselves.

PROVIZ is designed to include a graphical programming functionality which can be used to build a WSN application by assembling two, or more modules. These modules, which are required to build a sensor application, are called Simple Modules, and they are represented by a unique geometrical figure in the user interface. All the simple modules implementing a similar functionality, e.g. routing, can be grouped and represented using the same geometrical figure. Hence, the grouped simple modules can be used interchangeably while building an application. The graphical programming functionality can abstract the underlying programming syntax and wiring overhead from the application developer, and thus, reduces the development efforts for building a WSN application.

Along with the graphical programming functionality, PROVIZ supports a script-based programming language called the M-Core Control Language (MCL) [8], which has simple user-friendly commands to simplify the programming tasks in WSNs. In the MCL, a single command takes care of wiring the various interfaces with the configuration and module files of the new application. Also, it has commands which can be used to: 1) create new M-Core services [8]; 2) create new application files; and 3) add definition to the newly created application and service files. This reduces the development efforts of the users, as the MCL script takes care of all the wiring and updates different files for creating a new service, or application.

PROVIZ's integrated monitoring and programming tool can be used for programming a sensor application and to observe its behavior in a live environment. Hence, PROVIZ can be used like a control panel for network visualization, which has enhanced programming capabilities compared to the existing tools.

1.1.1.2 SIMAGE: Wireless Code Dissemination

In this thesis, an energy-efficient and secure code dissemination protocol for WSNs, named *Secure and Link-Quality Cognizant Image Distribution* (SIMAGE) [11] is also developed. PROVIZ's programing tool uses the SIMAGE protocol for reprogramming the sensor nodes over-the-air. SIMAGE uses an adaptive packet-sizing technique to adapt the code dissemination packet size based on the link quality between the sensor nodes, which reduces the number of retransmissions and energy consumption. Also, SIMAGE provides security services like confidentiality using the energy-efficient RC4 [12] algorithm and integrity using the CBC-MAC provided by the CC2420 [13] transceiver module of the sensors. SIMAGE has been evaluated in a network of real sensors and the results shows that dynamic link quality adaptive packet-sizing technique reduces the retransmission bytes by 93% and the per page transmission time by 35% when compared to the existing code dissemination protocols [5, 6].

CHAPTER II

LITERATURE REVIEW

NetViewer [3] is a visualization tool used for visualizing and monitoring a real-time WSN. NetViewer is designed in such a way that it will work for any user-defined packet format, which is facilitated by the *Packet Format Setter* module in NetViewer. NetViewer has a *Information Translator* module to translate the packet data to useful information based on the user-defined packet format. NetViewer, then, visualizes the packet transmissions and displays the translated packet data either in a tabular format, or in the form of a graph. However, NetViewer cannot do an off-line WSN visualization (by parsing the commercial packet sniffer's packet trace file (PSD format)) and cannot visualize a data trace generated by an external WSN simulator. Also, NetViewer is a standalone visualization tool, which does not have the capability to wirelessly program sensor nodes.

Octopus [4] is an integrated monitoring, visualization, and control tool developed for WSNs. Octopus is a protocol independent tool, which gets information about the sensor nodes and the network topology to visualize and monitor real-time sensor nodes in a WSN. Also, Octopus provides *Control Operators*, which can reconfigure the network behavior by sending short request messages. Although, Octopus as a network controlling tool can reconfigure, or change the working methodology of a WSN, it does not have an integrated programming tool to develop, simulate, and completely reprogram a sensor node over-the-air. Before deploying the sensor nodes, Octopus requires the user to program the different behavioral codes in the sensor nodes and associate them with a specific request message type. Therefore, these behaviors, which can change the network or individual nodes behavior, can be triggered with the help of short request messages. Similar to [3], Octopus cannot do an off-line visualization and cannot visualize by using data from a WSN simulator.

NetTopo [2] is an integrated simulation and visualization framework, which is designed

for validating algorithms used in WSNs. NetTopo is essentially a simulator, which is integrated with a real testbed to run the WSN algorithms and validate them for both scalability and accuracy. NetTopo provides a visualization tool to visualize the topology used in the simulation environment and it updates the information about connections, sensed data of each node, etc. However, NetTopo is specifically designed for simulating and validating algorithms, and hence, NetTopo cannot do live, or off-line WSN visualization and cannot program a sensor node.

TOSSIM [1] is a WSN simulator, which can simulate a homogeneous wireless sensor network with sensors running a common application. TOSSIM provides a visualization tool called TinyViz [1], which can be used to visualize, control, and analyze TOSSIM simulations. TinyViz can visualize the sensor nodes and the network traffic in a WSN. It provides information about the data transfered between nodes, serial communication packets, LED states, and provides a control window to change the simulation parameters. However, TOSSIM is a simulator and cannot do off-line visualization, or interact with real sensor nodes.

In short, all the tools discussed above are a standalone visualization tool, or a programming tool, or a simulator. PROVIZ is unique in its approach by providing all these functionalities under a single platform, which can program a sensor node, simulate a WSN application, and visualize the WSN traffic.

CHAPTER III

PROVIZ DESIGN OVERVIEW

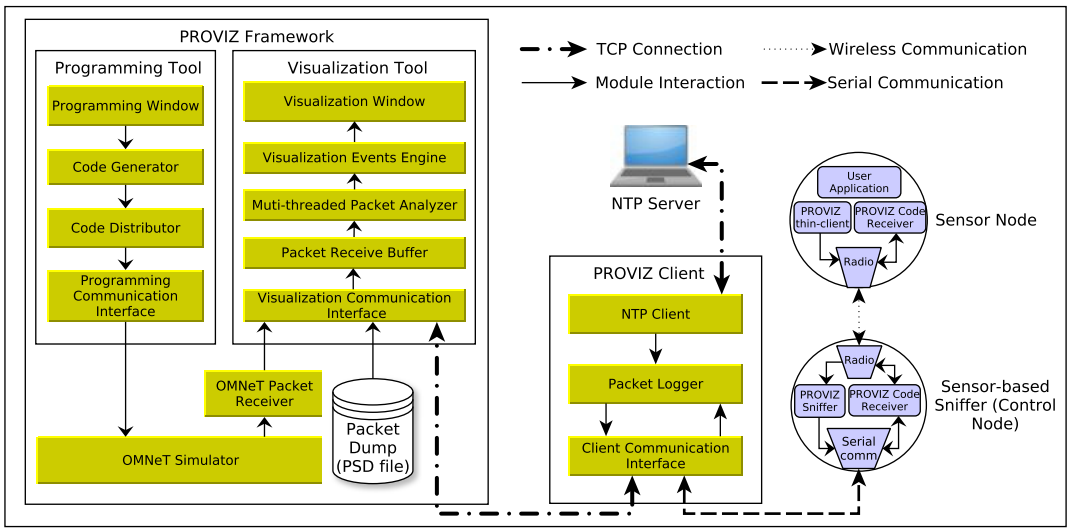

Figure 1: PROVIZ Architecture

PROVIZ is a multi-platform, modular, generic, extensible, and scalable programming and visualization framework [7] developed using the QT GUI Framework [14]. QT is a C++ based, open-source, and a cross-platform application and user interface development framework supporting most of the popular desktop and mobile operating systems [15]. Figure 1 shows the PROVIZ framework design, where PROVIZ runs in a host machine and has a *PROVIZ Client*, which can run in a local, or remote host and gather packet data using different sniffers. As shown in Figure 1, PROVIZ has a modular architecture, which enables the user to replace, or modify a module, without affecting the overall system behavior. PROVIZ provides two main functionalities. One is the functionality to visualize the data and the other is the functionality to program sensor nodes. For the former, PROVIZ can visualize the data either captured in real-time, or from a WSN simulator. The following sections discuss the design of PROVIZ programming and visualization functionalities and

explain the distributed model of the PROVIZ framework.

3.1 Modules for Programming Functionality

This section discusses the design of modules used for the programming functionality.

3.1.1 Programming Window

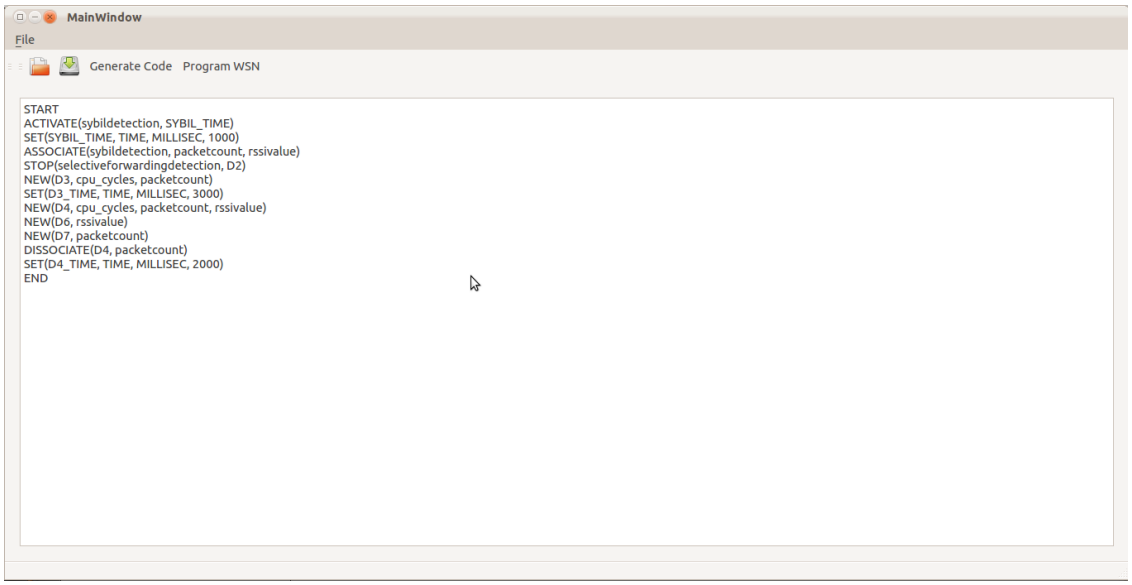

Figure 2: Screen Shot of PROVIZ Programming Window

The PROVIZ programming tool has an editor window, which can be used to edit the NesC [16] code of a TinyOS application, or a MCL [8] script. A screen shot of the PROVIZ editor window is shown in Figure 2. The MCL scripting framework provides user-friendly commands for creating new NesC components, interfaces, and wiring. The PROVIZ programming window is integrated with the visualization window in such a way that the user can switch between the tools by a single mouse click.

3.1.2 Code Generator

After entering the MCL script in the editor window, the user can use the *Generate Code* option (Figure 2) in PROVIZ to generate the NesC code from the MCL script.

3.1.3 Code Distributor

After generating the code, the user can modify, or change the implementation details in the NesC code and use the *Program WSN* option (Figure 2) to generate the WSN application binary. It uses the *Programming Communication Interface* to remotely program wireless sensor nodes by wireless code dissemination, which is discussed in detail in Chapter 6.

3.1.4 Programming Communication Interface

After the Code Distributor generates the code image, the Programming Communication Interface is utilized to disseminate the code image.

3.1.5 Wireless Code Dissemination

For wireless code dissemination, PROVIZ uses the SIMAGE protocol [11]. SIMAGE uses the Link Quality Indicator (LQI) value to asses the link quality between the nodes and then dynamically adapts the packet size. In a WSN having nodes with poor link quality, the dynamic packet-sizing technique reduces the number of retransmissions during code dissemination. Along with dynamic packet resizing, SIMAGE also provides energy-efficient security services, confidentiality and integrity provided by the CC2420 [13] transceiver module in sensors (e.g., MICAz, TelosB, etc.). A detailed description about the SIMAGE protocol is presented in the Chapter 6.

For programming a WSN, PROVIZ uses the *Control Node* as a base station for disseminating the code image. All the nodes in the WSN along with the user application should have a *Code Receiver* module, which can receive the disseminated code image to program the sensor node and transfer it to the neighboring nodes. The Control Node is also utilized in the visualization functionality as a sniffer as explained in the next Sub-section.

3.2 Modules for Visualization Functionality

This section discusses the design of modules for the visualization functionality and Figure 3 shows the working of the modules in PROVIZ visualization tool.

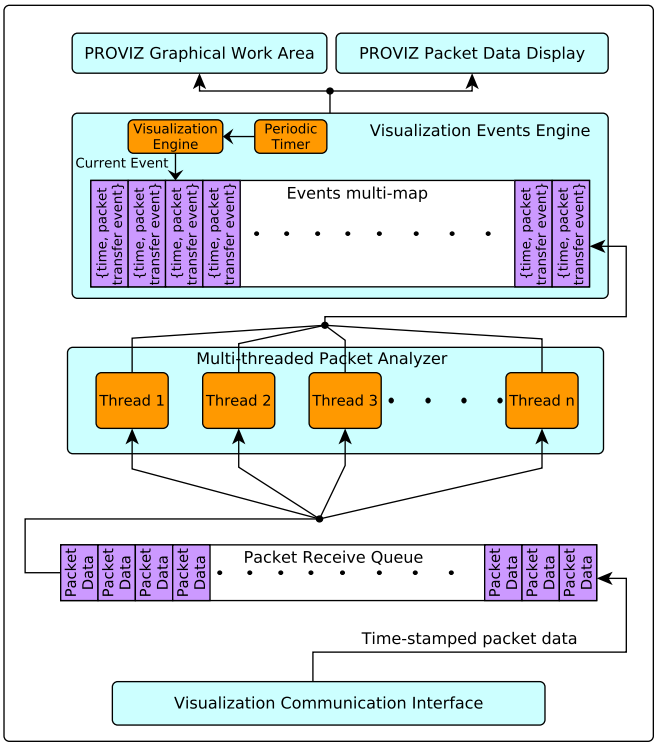

Figure 3: Illustration of PROVIZ Visualization Tool

3.2.1 Visualization Communication Interface

This interface has a generic packet receive module for the visualization functionality, which can receive packets from different sources such as: a 1) PROVIZ Client with a sensor-based sniffer attached to either a remote, or a local host; 2) WSN simulators such as OMNeT [9]; and 3) Packet trace file, generated by commercial sniffers, e.g., the PSD [10] file format created by the TI SmartRF packet sniffer [10].

3.2.2 OMNeT Packet Receiver

While simulating the WSN application in the OMNeT simulator, the OMNeT radio receive driver outputs the time-stamped packet trace. The OMNeT Packet Receiver fetches the network simulator trace and sends it to the Visualization Communication Interface. The detailed working of OMNeT Packet Receiver is discussed in Section 4.2.

3.2.3 Packet Receiver Buffer

The Visualization Communication Interface receives the packet data and stores the data in the form of an byte-array in the Packet Receiver Buffer queue, which can be fetched later for processing.

3.2.4 Multi-Threaded Packet Analyzer

While capturing the packets, PROVIZ initiates a multi-threaded module, which parses the packet data stored in the Packet Receive Buffer queue. The multi-threaded module parses, for the packet arrival time, IEEE 802.15.4 header information [17], and the packet payload, and then it matches the packet data with the user-defined packet formats. Since the Packet Analyzer has to parse all the incoming packets and translate them, it becomes a computation intensive operation. To achieve faster processing and avoid bottlenecks, the Packet Analyzer module is designed in a multi-threaded fashion. The number of threads running in the Packet Analyzer module is a configurable parameter, which can be modified by the developers based on the incoming packet arrival rate. Whenever the application starts, based on the value in the thread count parameter, PROVIZ creates sufficient amount of worker threads and forms a thread-pool. So, whenever the user starts a visualization, the worker threads from the thread-pool is used to parse the packets, which eliminates the delay in thread creation.

3.2.5 PROVIZ Visualization Events Engine

After parsing the packets, the Packet Analyzer module creates a packet transfer event and adds it to the *PROVIZ Visualization Events Engine*, which has a multi-map data structure with time of the event as key. The Events Engine multi-map includes all the events in sequence, sorted based on the time of the event. To make the Visualization Engine extensible, PROVIZ defines an abstract class with event handler methods as pure virtual functions. So, if users want to extend PROVIZ with new features, they simply need to inherit the abstract class and define the event handler methods, which can implement features like node mobility, battery-level monitoring, etc. Then, the new event objects can

be added in the Events Engine multi-map and these event handlers are then called by the Events Engine at the appropriate time. When the visualization starts, the time of the first event in the event queue is considered as the current visualization time and the Events Engine starts a periodic timer. Whenever the timer is triggered, the current visualization time is incremented and the events that are to be executed at this current visualization time are identified in the multi-map and they are triggered by calling their event handlers.

After starting the visualization, PROVIZ provides a graphical canvas (Figure 6) called *PROVIZ Graphical Work Area*, for visualizing the sensor nodes and packet transfers. Also, it includes a *PROVIZ Packet Data Display Window* (Figure 9), which shows the parsed packet information in a user readable format. These are discussed in further detail in Chapter 4.

3.2.6 PROVIZ Client Design

PROVIZ provides a *PROVIZ Client* application running in local, or multiple remote machines for gathering WSN packets. The Control Node is connected to these remote/local machines running the PROVIZ Client. This node works as a sniffer for gathering WSN traffic while also serving as a base station for disseminating the code image over-the-air.

3.2.6.1 Client Communication Interface

The Client Communication Interface receives the sniffed packets from the Control Node and the *PROVIZ Packet Logger* module adds a time-stamp to the packet data. The time-stamped packet data is sent to the visualization host over the network using a TCP socket.

3.2.6.2 Network Time Protocol Client

To synchronize the data collection and visualization in a distributed environment, the PROVIZ Clients are time-synchronized with the help of the Network Time Protocol (NTP) [18]. The NTP Client requests the global time from the NTP server and then synchronizes the system time based on the server's reply.

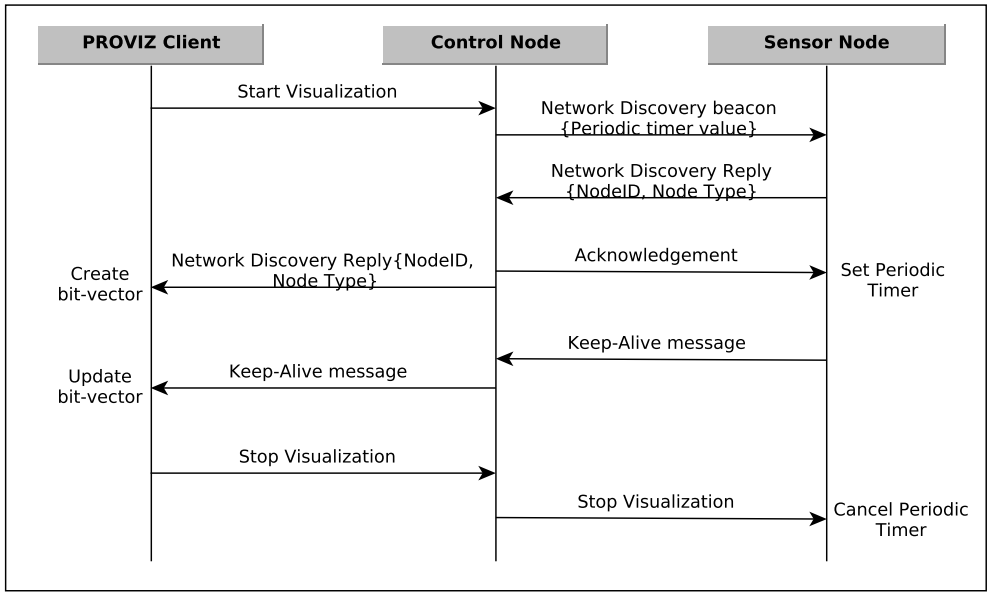

Figure 4: PROVIZ Network Discovery Module

3.2.7 Thin-Client and Network Discovery

PROVIZ provides an optional thin-client module that resides in sensor nodes along with user programs. Also, PROVIZ includes a Network Discovery process in the PROVIZ Client, which is used to find the number of sensor nodes available in a WSN and their type. Figure 4 illustrates the procedure of network discovery in PROVIZ. When the PROVIZ Client starts, it triggers the Control Node to send a network discovery broadcast message. All the sensor nodes running the *PROVIZ thin-client*, on receiving this message, responds back to the Control Node with a specific message. This message includes the Node ID and the type (e.g., MICAz, TelosB) of the sensor. After receiving the reply, the Control Node sends an acknowledgment and sends the received data to the PROVIZ Client. The PROVIZ Client identifies the unique Node IDs and creates a *Keep-Alive bit-vector* of appropriate size. Also, it reports the number of sensor nodes and their types to the PROVIZ visualization host. The thin-client, after getting an acknowledgment from the Control Node, starts sending a periodic (e.g., every 10 seconds) keep-alive message to the Control Node. The periodic interval in which the keep-alive message has to be transmitted, is sent with the network

discovery broadcast message. The PROVIZ Client, on receiving a keep-alive message, updates the Keep-Alive bit-vector based on the source Node ID. At the end of the periodic interval, the PROVIZ Client checks the bit-vector for nodes, which did not send a keep-alive message in that round and then clears the bit-vector before proceeding to the next round. If a node fails to send keep-alive messages continuously for a specified number of rounds, the PROVIZ Client communicates the node IDs to the visualization host, which in turn notifies the user. Then, PROVIZ decrements the number of live sensors that can be used for visualization. When the user closes the PROVIZ Client, it triggers the Control Node to send a broadcast message to stop sending the periodic keep-alive messages. The PROVIZ Client can do the network discovery for WSNs without thin-client, by analyzing the sniffed packets and creating a bit-vector based on this information. However, in this case PROVIZ cannot get the sensor state information. Thus, PROVIZ can still work with WSNs without thin-client, but with reduced functionalities.

3.3 PROVIZ Distributed Framework Model

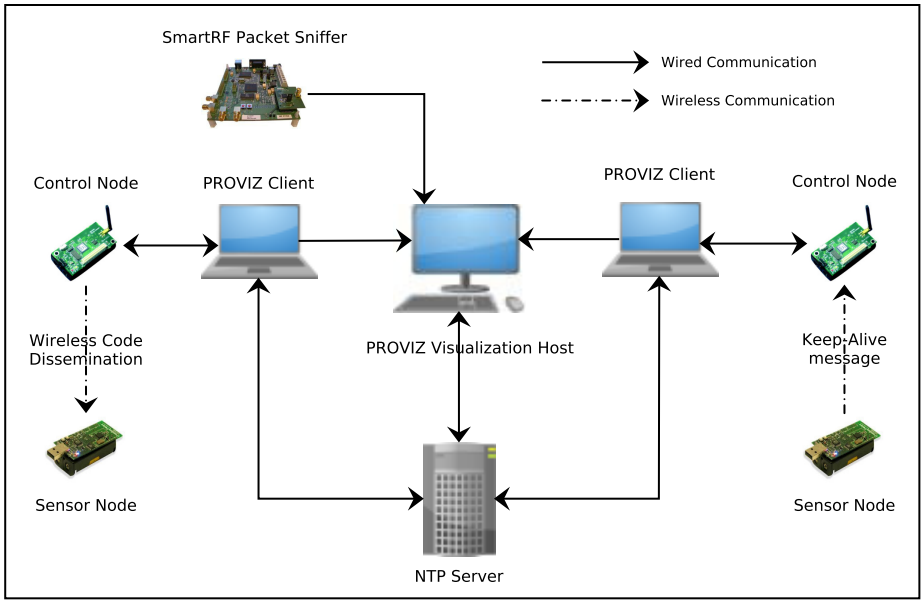

Figure 5: PROVIZ Distributed Framework Model

For visualizing a large WSN, PROVIZ uses a distributed approach that uses multiple

sniffers as shown in Figure 5. In the distributed setup, the PROVIZ visualization tool runs in a host machine and multiple sensor-based sniffers are placed in a distributed fashion so that a packet transmitted by a sensor node can be sniffed by at least one of the sniffers. The PROVIZ Client program runs in a time-synchronized remote, or local host connected to a sniffer and sends the time-stamped packet data to the visualization host for visualization. In a distributed setup, some nodes will be in the sniffing range of multiple sniffers and hence, PROVIZ can receive duplicates of a same packet. However, since the PROVIZ Clients are time-synchronized, PROVIZ Events Engine can identify and remove the duplicates by validating the time-stamp and packet data.

CHAPTER IV

VISUALIZATION TOOL FEATURES

This chapter discusses the various features of PROVIZ visualization functionality.

4.1 PROVIZ User Interface

Figure 6 shows the screen shot of the PROVIZ visualization tool with a group of infrastructure monitoring sensors sending structural health information [19] periodically to a cluster head. The PROVIZ Visualization User Interface window has a: 1) *Control toolbar*, which provides the control buttons for the visualization; 2) *Drag and Drop Window Holder*, which has sub-windows to hold the images of sensor nodes that are available in a WSN and to hold the demo application icons; and 3) *Graphical Work Area*, a canvas where the sensor node images are placed and the packet transfers are visualized.

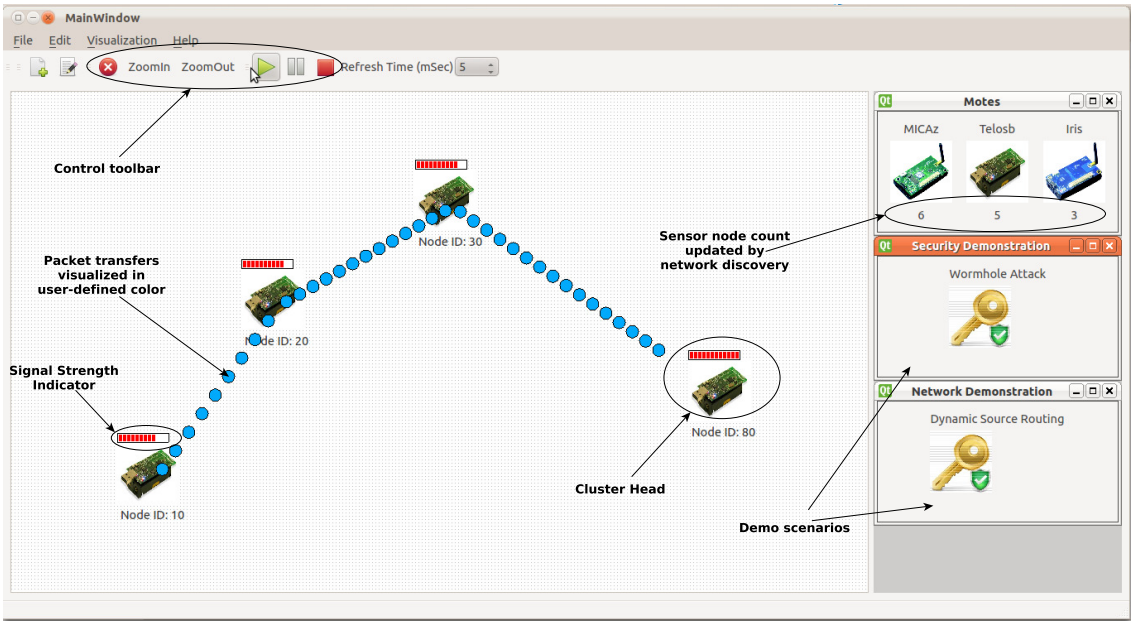

Figure 6: Screen shot of PROVIZ Visualizing an Infrastructure Monitoring WSN

4.1.1 Control Toolbar

The Control toolbar provides control options to start, pause, and stop the visualization. Also, it has support to zoom-in, zoom-out, and clear the nodes in the Graphical Work Area.

4.1.2 Drag and Drop Window Holder

The Drag and Drop Window Holder has multiple sub-windows and these sub-windows can hold either sensor node icons, or icons associated with demo applications. The sensor icons displayed in the sub-window include an image of sensor node and a count associated with them. The count depicts the number of sensors of that type available in a WSN and it is determined by the PROVIZ Network Discovery module (Section 3.2.7) in the PROVIZ Client. PROVIZ uses this count value associated with each sensor icon to restrict the number of sensors that can be visualized. To visualize a WSN, the sensor icons in the sub-windows can be dragged and dropped into the Graphical Work Area. Each node that is dropped, needs to be associated with a unique Node ID, which is displayed along with the sensor icon. Also, a signal strength meter showing the signal strength of a node is displayed at the top of each sensor node icon. The signal level value is received from sensors in real time as explained below.

4.1.3 Packet Visualization

The Control Node sniffs the packets complying with IEEE 802.15.4 and transmits the header and payload information of packets to the PROVIZ Client through serial communication. The Control Node, having CC2420 transceiver module [13] can estimate the wireless link quality and give a Link Quality Indicator (LQI) value. The sniffer program in the Control Node fetches the LQI and RSSI value information from the CC2420 transceiver and appends it to the packet payload before sending the sniffed packet data to the PROVIZ Client. The PROVIZ Client sends this data to the visualization host and the multi-threaded Packet Analyzer (Section 3.2.4) in the visualization host parse the packets, for LQI value and associate it with the packet transfer event, which is enqueued in the PROVIZ Visualization

Events Engine (Section 3.2.5). When the packet transfer event handler is triggered, the signal strength meter of corresponding node is updated with the new value. On receiving the packets from the serial interface of Control Node, the PROVIZ Client prepends the time-stamp to the packet data and sends it to the visualization host. Utilizing this packet information format, starting with time-stamp, followed by packet data, RSSI, and LQI, which is used by the popular SmartRF packet sniffer [10], ensures that PROVIZ is generic and extensible. This feature is also utilized to visualize the packet transfers simulated by the OMNeT simulator, with the help of *OMNeT Packet Receiver*. For this, the TinyOS communication driver for the OMNeT simulator was modified to output the packet information whenever a node receives a packet and the OMNeT simulator creates a packet trace with this time-stamped packet information. The OMNeT Packet Receiver module reads the packet information and converts it to the SmartRF packet sniffers packet format, which can be parsed by the PROVIZ Packet Analyzer.

4.2 Heterogeneous WSN Visualization

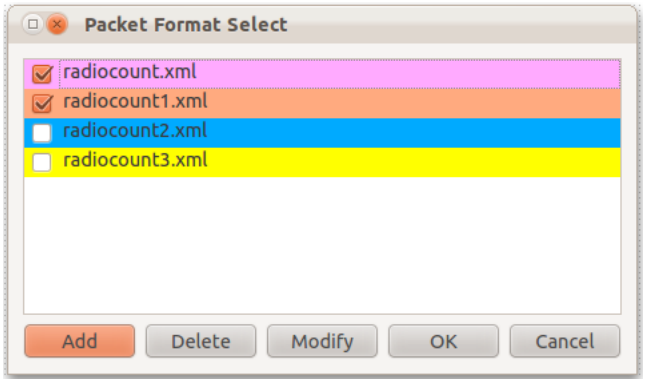

Figure 7: PROVIZ Packet Formats Selector Window

Most WSNs are heterogeneous with different types of sensor nodes sending packets with different packet payload formats. PROVIZ is capable of visualizing such a heterogeneous network and it can understand different packet formats. To select multiple packet formats for visualization, PROVIZ provides a *Packet Format Selector* window (Figure 7), which lists the user-defined packet formats for the users to select. Also, PROVIZ has a feature to define the packet format graphically using the *PROVIZ Packet Format Specifier* as shown

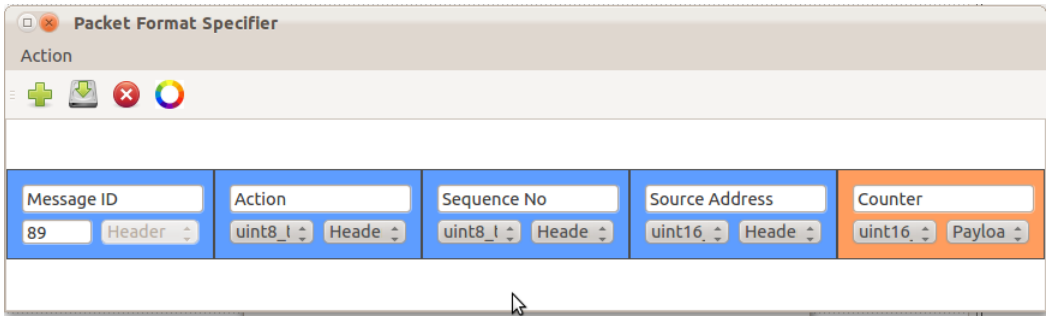

Figure 8: PROVIZ Packet Format Specifier Window

in Figure 8. For defining a packet format, PROVIZ expects the users to start the packet format with a unique message ID, which is used by the PROVIZ Packet Analyzer to identify and match the packet data to a particular packet format. Then, the user can add multiple packet fields and specify the data-type for the field and the packet field name for each of the packet fields. After defining the packet format, PROVIZ expects the users to select a packet animation color using the color selection tool. The packets matching this format are visualized with the color specified by the user. The user can then save the color information along with the packet format details in an XML file, which can be exported or shared with other PROVIZ users. Also, users can define the packet format in the form of an XML code directly and give that as input to PROVIZ. A sample packet format XML code snippet is shown in Listing 4.1.

Listing 4.1: Packet Format XML

```
<Packet>
  <msgID>90</msgID>
  <Color>
    <r>0</r>
    <g>170</g>
    <b>0</b>
  </Color>
  <Header index="1">
    <Description>Action</Description>
    <Datatype>uint8_t</Datatype>
```

```
  </Header>
  <Header index="2">
    <Description>Sequence No</Description>
    <Datatype>uint8_t</Datatype>
  </Header>
</Packet>
```

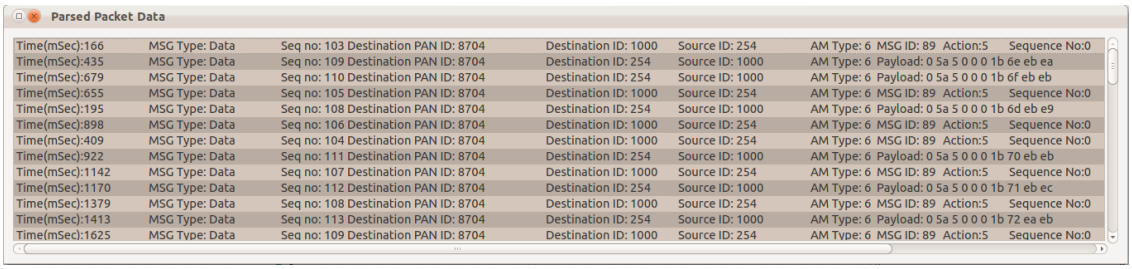

Figure 9: Screen Shot of PROVIZ Packet Display Window

By default, the Packet Analyzer can understand and translate the IEEE 802.15.4 header information [17] in the input packet data. After interpreting the header information, the Packet Analyzer matches the raw packet payload data with a user-defined packet payload format based on the message ID value. After finding a match, the Packet Analyzer extracts the necessary bytes and type-casts it to the data-type mentioned in the packet payload format. Similarly, the rest of the packet data is type-casted to the appropriate data-types specified by the packet fields in the packet payload format and it is displayed along with the packet field description in the *PROVIZ Data Display* (Figure 9). If the Packet Analyzer cannot find a match with a packet format, it displays the hex value of the payload in the Data Display window. Figure 9 has the translated packet information of packets matching the packet payload format shown in Figure 8 and also shows the hex value of the packet payload, whose format does not match with any of the user-defined packet payload format.

4.3 Demo Scenario Visualization

PROVIZ includes built-in extensible visual demo deployment scenarios, on which users can click and visualize easily as shown in Figure 6. Users can use this feature to create a demo

scenario to visualize a critical/complex WSN deployment and share it with other PROVIZ users. The demo visualization can be developed by creating an XML code, which specifies the node type to be used, number of nodes, node ID, node location, and the time, when the packet transfers should happen. To define a packet transfer, each transfer is considered as a frame, which has the information such as the node IDs between which the transfer should happen and the time when the transfer should be initiated. A sample demo visualization XML snippet is shown in Listing 4.2.

Listing 4.2: Demo Scenario Visualization XML

```
<Example_Scene>
  <Name>Packet Transfer Example</Name>
  <Node>
    <Node_Count>1</Node_Count>
    <Mote Node_ID="0">
      <Position>
        <x>-400</x>
        <y>0</y>
      </Position>
      <Description>Basestation</Description>
      <Node_Type>Telosb</Node_Type>
      <Mote_Image_Path>images/telosb.jpg
      </Mote_Image_Path>
    </Mote>
  </Node>
  <Simulate>
    <Frame time="100">
      <Start>0</Start>
      <End>10</End>
       <Color>
         <r>0</r>
         <g>0</g>
         <b>255</b>
```

```
        </Color>
       <SignalStrength>100</SignalStrength>
     </Frame>
   </Simulate>
</Example_Scene>
```

CHAPTER V

PROGRAMMING TOOL FEATURES

This chapter explains the graphical and script-based programming functionalities of PROVIZ in detail.

5.1 PROVIZ Scripting Framework

PROVIZ supports the MCL, a script-based programming language developed in python, for creating WSN applications in NesC [16] code. The PROVIZ Scripting framework provides simple commands, which abstracts the wiring concept of NesC programming language from the user. This enables advanced application developers to focus on the programming logic instead of learning the NesC programming language. Figure 10 shows sample MCL code, which generates the wiring code for the new D4 module and updates the configuration file.

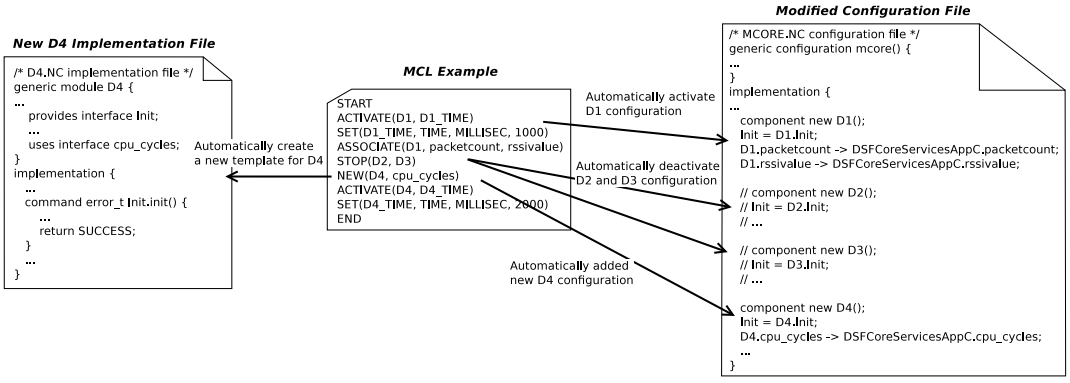

Figure 10: Example Usage of MCL

Adapted from [8]

5.2 Design Description of Graphical Programming Tool

This section describes the design of the PROVIZ Graphical Programming functionality. PROVIZ is designed to include graphical programming functionality, which considers a

WSN application as a *Complex Module* built by assembling two, or more *Simple Modules*. The Simple Modules are small, standalone programs implementing a basic operation in a sensor node (e.g., a program for gathering temperature sensor value, or for implementing a timer, or for implementing a routing function in sensors, can be considered as Simple Modules). Simple Modules implementing a similar operation, e.g. routing operation, are grouped, and all the Simple Modules in a group can have the same unique geometric shape and displayed in a same sub-window in PROVIZ. The Simple Modules grouped together should expose the same interface, but, the implementation of the commands and the events can be different. For example, the Simple Modules grouped under the routing group exposes a common interface, which has a command called *routeAndSend()* and an event *sendDone()*. The command *routeAndSend()* takes care of determining the route to the destination and sends the packet to the next hop node, but, the implementation differs between the Simple Modules depending on the routing protocol used.

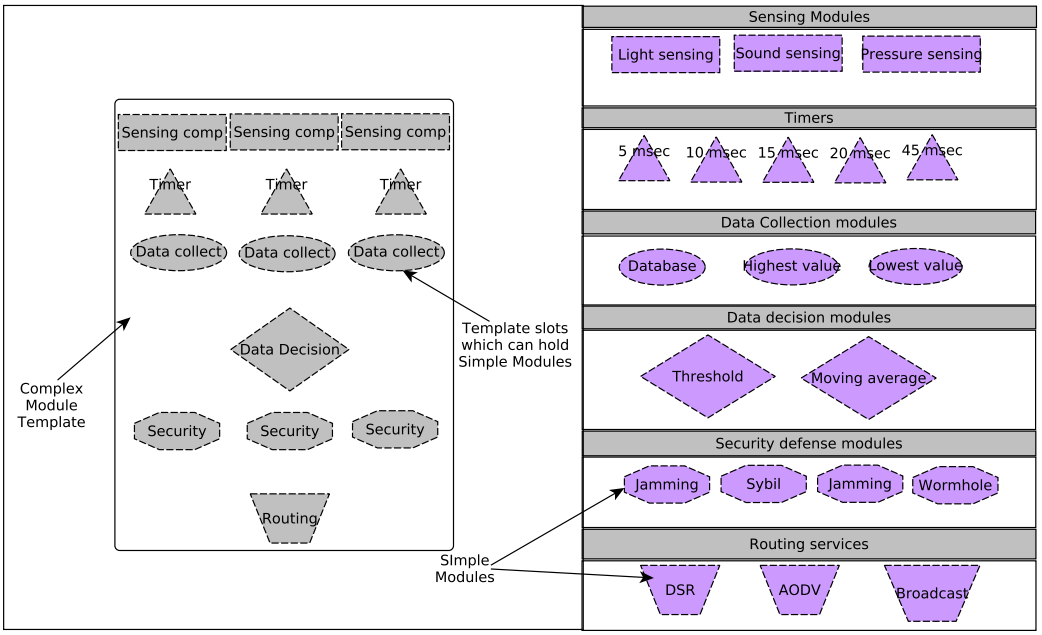

Figure 11: Sample Illustration of PROVIZ Programming Tool User Interface

In the user interface illustration shown in Figure 11, the Complex Module is represented in a rectangular template, which has slots to hold the necessary Simple Modules. The slots in

the template signifies that the Simple Modules from the appropriate groups are required to make WSN application complete. These slots are of the same shape as the corresponding Simple Modules, which it can hold. This helps a novice programmer, whose job is now reduced to dragging and dropping of the appropriate shapes into the template slots, to build a sensor program. The template representing the Complex Module is associated with a program, which implements the algorithm to process the input data gathered from the Simple Modules and to feed appropriate inputs to the Simple Modules. Once the Complex Module is assembled completely, the PROVIZ Graphical Programming tool should create an XML code segment having the information about the Simple Modules constructing the Complex Module, and the XML code can be given as input to the Sensor Code Generator. The PROVIZ Graphical Programming tool by default should provide support for some of the commonly used Simple Modules and templates. Also, it should provide flexibility for the user to add a new Simple Module, or template, or even extend the existing Simple Module and template.

In PROVIZ, an XML code segment is generated to communicate the programming details to the Code Generator, which interprets the XML code segment and generates NesC code. Since, an XML code segment is given as input, the user can build his/her custom Code Generator to interpret the XML code segment and generate WSN application code for different programming languages. This makes the PROVIZ programming functionality more generic, and extensible for user requirements. The full implementation of the PROVIZ visual programming language will be a part of our future work.

CHAPTER VI

SIMAGE: A WIRELESS CODE DISSEMINATION PROTOCOL

WSNs are used in a range of critical domains where it is necessary that the nodes be reprogrammed with a new, or modified code image without removing them from the deployment area. Various protocols have been developed for the dissemination of code images between sensors in a multi-hop WSN, where these sensor nodes may have varying levels of link quality. The code dissemination process in these protocols is hindered by the nodes with poor link quality. This results in an increased number of retransmissions and code dissemination time. Also, in several of the techniques, the code dissemination process is not secure and can be eavesdropped, or disrupted by any malicious wireless sensor node in the transmission range. To overcome these overheads a simple approach is proposed, called *Secure and Link-Quality Cognizant Image Distribution* (SIMAGE) [11], to enhance the existing code dissemination protocol using the available resources in the sensors. This approach, link-quality based dynamic packet-sizing, aims to reduce the number of retransmissions and overall code dissemination time. This approach also provides confidentiality and integrity to the code dissemination process by utilizing energy-efficient encryption methods like RC4 and hardware authentication methods like CBC-MAC. SIMAGE is evaluated in a network of real sensors and the results shows that dynamic link quality adaptive packet-sizing technique reduces the retransmission bytes by 93% and the per page transmission time by 35% when compared to the existing code dissemination protocols. The SIMAGE code dissemination algorithm and the trade-offs between reliability, security overhead and overall transmission time are discussed in detail in the below sections.

6.1 Motivation

Sensors are deployed in various locations such as underwater, volcano regions and underground, making them hard to recover, or replace. Due to this diverse deployment of Wireless

Sensor Networks (WSNs) and the multifarious functionalities provided by them, reprogramming the deployed sensor nodes through wireless links becomes a necessary and desirable task. For instance, if software running on the node requires an update as a result of a security patch, or additional functionality, it is necessary to replace the existing code on the node with the new updated code. This process of disseminating code over wireless links is called *code dissemination*. Under hostile conditions, an attacker might try to modify, or get access to a disseminated code. The attacker could also attack the dissemination process itself by injecting malicious code dissemination packets to exhaust the limited energy of the sensor nodes. Therefore, it is imperative that the dissemination process be secured.

In addition to security, code dissemination protocols should address the variation in link quality between nodes. The nodes with poor link quality hinders other nodes in the network from proceeding forward in the code dissemination process. Moreover, such nodes with poor link quality yields increased retransmissions, which further degrades the security of the system, as an attacker has multiple opportunities to intercept the disseminated packet. To overcome these overheads a *Secure and Link-Quality Cognizant Image Distribution* (SIMAGE) is introduced for WSNs, which aims at reducing the number of retransmitted code dissemination packets along with providing security services such as confidentiality using RC4 [12] encryption and integrity using the CBC-MAC provided by the CC2420 [13] transceiver module of the sensors (e.g., MICAz and TelosB). The results show that a network of MICAz sensors are more efficient while using SIMAGE for code dissemination than other existing code dissemination protocols, in terms of code dissemination time and total number of retransmissions encountered. From the experimental results it is observed that, SIMAGE reduces the retransmission bytes by 93% and the per page transmission time by 35% when compared to the existing code dissemination protocols.

The contributions of this work are three-fold: 1) designed and implemented a secure code dissemination protocol - SIMAGE, which 2) takes into account link quality between nodes during the dissemination process, dynamically adapts the packet size based on link quality, that 3) enables the scheme to outperform previously proposed techniques.

6.2 Related Work

Deluge [5] is the most widely used code dissemination protocol and is included in the recent TinyOS distributions [20]. Deluge is a page-by-page code dissemination protocol proposed for WSNs. The base node advertises the new version of code and the receiver node in response sends a request message, for the new version. Next, the base station broadcasts the first new page of the code image to the receiver nodes, with fixed sized data packets. Once all receiver nodes have received all the packets corresponding to that page, the next page transmission begins. In case a receiver node has lost some packets in the first page, all the lost packets are retransmitted prior to the transmission of the next page. The code dissemination is not secured and can be eavesdropped by an attacker. Moreover, as Deluge does not consider the link quality between nodes, the retransmissions between nodes with poor link quality severely delays the propagation of the code image.

The Secure Network programming protocol [21] uses a public-private key encryption to sign the advertisement packets of the code image and SHA-1 for computing the hash of each packet. In this work, the computed hash of one future packet is embedded into the payload of the previous packet in sequence and the hash of the first packet is embedded into the advertisement packet, which is signed. So, a receiver node, upon authenticating the advertisement packet, verifies the integrity of the next received packet immediately and saves, or discards the packet based on the computed hash. However, in the case of out of order delivery, the receiver needs to cache the packet and wait for the previous packet in sequence to verify the cached packets. This can be used by the attacker to inject fake packets and deplete the cache memory of the receiver node and cause denial of service attacks.

Seluge [6] is essentially a secure version of Deluge and therefore it preserves the page-by-page propagation method of Deluge. Similar to [21], a hash of each packet in a page is computed using SHA-1 and this value is embedded in the correspondingly sequenced packet of the previous page. So, once all the packets in a page are received, the receiver has all the hash values for the packets in the next page. Seluge also uses the *Elliptic Curve Digital Signature Algorithm* (ECDSA) to sign the initial code image advertisement. However, the

SHA-1 and the ECDSA algorithm used are time consuming processes ([22], [23], [24]). Note, [21] and [6], similar to [5], do not take into account the link-quality between nodes in the network.

Another recent work on the code dissemination process in WSNs, DiCode [25], provides a secure and distributed code dissemination protocol. As opposed to the aforementioned centralized approaches ([5], [21], [6]), the key difference in this technique is the usage of distributed control for code dissemination. However, the security features used in this approach are similar to that of Seluge. The experimental results also indicate that DiCode has longer propagation delays when compared to Deluge and Seluge. Although [25] proposes a distributed code dissemination protocol, it does not address any of the inter-node link quality issues.

Nonetheless, as discussed in [26] with experimental results, link quality prediction is important for better system provisioning and resource management. Moreover, the experiments conducted on IEEE 802.15.4 Zigbee radios with CC2420 chipset illustrate that the *Link Quality Indicator (LQI)* is linear with respect to the instantaneous SNR. Finally, they assert that LQI can be used as an effective parameter to predict instantaneous link quality between nodes [26].

Therefore, an enhanced protocol named SIMAGE is proposed, which provides secure and link-quality aware code image dissemination for WSNs.

6.3 Background Work

A common problem in many wireless networks is varying and poor link quality of the channel between the communicating nodes [26]. During communication, a reduction in the link quality increases the number of retransmissions between the nodes to transfer a useful information. The increased retransmissions have a significant impact on WSNs, which consists of resource-constrained sensor nodes. The code dissemination protocol in TinyOS [5] uses a fixed packet size to transmit the code image between the wireless sensor nodes and is susceptible to increased packet loses under poor link quality conditions. For the same code image size because of the increased packet loss rate, the code dissemination protocol

in TinyOS [5] needs more retransmission bytes to disseminate the code image, as analyzed in Section 6.7. This, in turn increases the total code dissemination time between the nodes.

The code dissemination protocol included in TinyOS distributions is a page-by-page transmission protocol, where a code image is split into pages of size 1024 bytes. Each page is divided into an equal number of packets with a packet size value set by the user. If the base station has a newer version of the code image, it advertises this information to the network. Upon receiving the advertisement, nodes in the network send a request for a particular page. If there is packet loss during the transfer of the page from the base station, the receiver makes a request for the lost packets until all of the packets are received. For instance, consider that there are two receiver nodes, which are one hop away from the base station. The first node has a channel with good link quality and the second node has a channel with poor link quality. Then, during the dissemination process from the base station, the first node receives all the packets with less retransmission rounds when compared to the second node. The first node, before sending a request message for the next page, has to wait until the second node receives all the packets of the current page. Thus, the code dissemination time for the entire network is affected by the increased retransmissions of the node with the poor link.

To analyze this, a simple preliminary experiment is conducted to calculate the packet retransmission rate for various packet sizes with different link quality conditions between a pair of MICAz motes. The link quality of the channel is estimated using the LQI value in the MICAz mote.

Using the above experimental setup, the first mote transmits 50 packets and the number of retransmissions is calculated. The experiment is repeated 10 times, with the same packet size and link quality condition. The average retransmission per 50 packets is calculated and the average Packet Retransmission Ratio (ψ) is calculated with the average retransmission value using Equation 1, where, φ is the number of retransmitted packets and τ is the total number of packets transmitted, including the retransmitted packets.

$$\psi = \frac{\varphi}{\tau} \tag{1}$$

The procedure is repeated to calculate the ψ value for various packet sizes and various link quality conditions. Figure 12 shows the variation of ψ value for increase in packet size value for four different LQI ranges. The graph shows that for a poor LQI range, the value of ψ worsens as the packet size increases and becomes nearly equal to one. For the average LQI range, the value of ψ shows an increasing trend with an increase in packet size. For the good and best LQI ranges, the value of ψ remains nearly equal to zero, for a packet size increase until 45 bytes. However, beyond 45 bytes, the value of ψ for the good LQI range increases steadily while the ψ value still remains zero for the best LQI range.

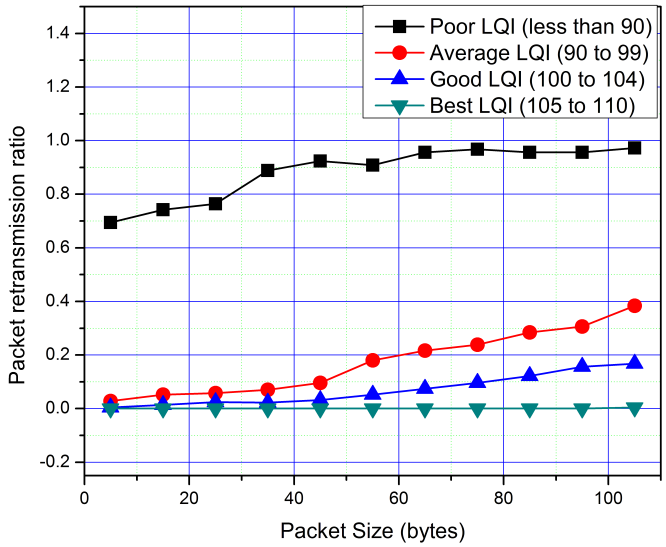

Figure 12: Packet Retransmission Ratio (ψ) vs Packet Size

Using the value of ψ, computed from the experiment for various packet sizes and LQI values, the number of packets retransmitted per page can be calculated using Equation 2.

$$\phi = \sum_{i=o}^{\gamma-1} (\psi * n)(\psi)^i \tag{2}$$

$$\frac{d(n * \psi^{(\gamma+1)})}{d\gamma} = 0 \tag{3}$$

where, ϕ is the cumulative number of retransmitted packets per page, γ is the total number of retransmission requests needed to transmit the entire page of size 1024 bytes,

which can be computed using the Equation 3. In Equation 3, γ is the minimum integer value, starting from zero, that satisfies the equation and n is the number of packets per page.

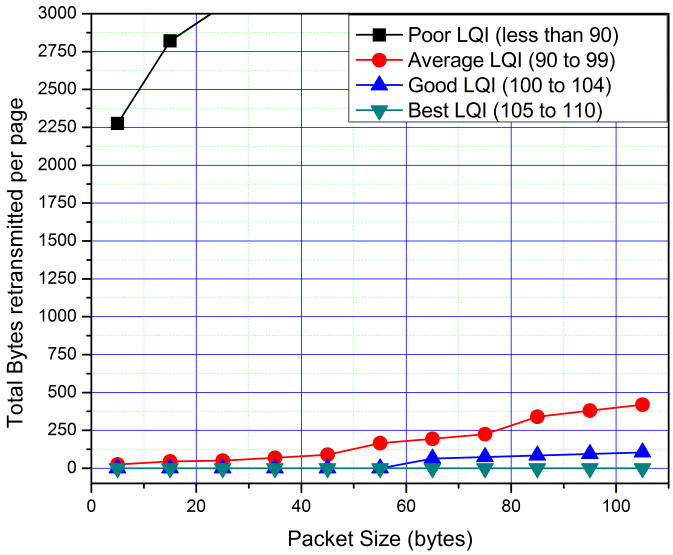

Figure 13: Number of Bytes retransmitted per page vs Packet size

Figure 13 shows the number of retransmitted bytes per page in the code dissemination protocol for various packet sizes, in different LQI ranges. The figure shows that, for an increase in packet size up to 55 bytes, the retransmitted bytes per page remains zero for the good and best LQI ranges. However, when the packet size exceeds 55 bytes, the retransmissions increase gradually for the good LQI range, while it remains zero for all packet sizes in the best LQI range.

Since transmitting a page with minimum retransmissions reduces the code dissemination time, to achieve better code dissemination under fluctuating link quality conditions, a LQI based adaptive packet size code dissemination protocol is proposed. This protocol samples the LQI of all the receivers and determines the optimal packet size for disseminating a page. The details of the scheme is explained in Section 6.4.

6.4 Proposed Scheme

SIMAGE, which builds upon the code dissemination protocol of TinyOS, Deluge [5], was implemented and tested in a network of MICAz sensor motes. This approach has two specific features, first is the dynamic, adaptive packet size estimation, which determines an optimal packet size for transmitting a page using the LQI values of the request messages. Second, it provides security for code dissemination with stream cipher RC4 encryption and integrity using CBC-MAC. The details of the implementations are explained below.

6.5 Dynamic Adaptive Packet Size Algorithm

Table 1: Optimal payload size for different LQI ranges

LQI	**Value Range**	**Optimal Payload Size**
Poor LQI	< 90	5
Average LQI	90 - 99	20
Good LQI	100 - 104	40
Best LQI	105 - 110	80

The experiments in Section 6.3 illustrate that different LQI ranges have different optimal packet sizes. So, using an adaptive packet size in the code dissemination process is instrumental to reduce the per page dissemination time. Hence, from the experiment in Section 6.3, the optimal data packet size values for different LQI ranges have been determined and are shown in the Table 1. The data packet size in the table does not include the 7 byte data header overhead and the security header overhead.

The link quality between nodes in the network is measured using the LQI provided by the CC2420 transceiver module of the sensors. When the base station receives the request message, it stores the source ID and the LQI values of the requester node. While sending the first data message, the base station computes the average LQI of all the requester nodes and this gives a estimation of the link quality of the channel around the base station. Based on the measured average LQI, the size of the packets in the corresponding page is dynamically adapted to reduce the number of retransmissions involved. The algorithm for adaptive packet-sizing used in the receiver and transmitter nodes are shown in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 Receiver Node

for *each received data message* **do**
  if $DataPktSize \mathrel{!}= CurrPktSize$ **then**
    $modify\_parameters()$
  end if
end for

function $modify\_parameters()$
  $\delta \leftarrow DataPktSize/CurrPktSize$
  $CurrPktSize \leftarrow DataPktSize$
  $NewBitvectorSize \leftarrow CurrPktSize/\delta$
  $NewBitvector \leftarrow Bitvector(\delta)$
end function

In the transmitting node, if it is a first round of data transmission for a page, the *moving average (MA)* is calculated as a function of the *average LQI (ρ)* and *scaling factor (sf)* as following,

$$MA = \rho * sf + (1 - sf) * MA \tag{4}$$

Algorithm 2 Transmitter Node

for *each request message receieved* **do**
  $lqi \leftarrow LQI(RcvdPkt)$
  $map\langle src\_id, lqi\rangle$ $lqi\_map \leftarrow (SrcId, lqi)$
end for
for *first data message transmission* **do**
  if *retransmission* **then**
    compute $\rho(lqi\_map)$
    $DataPktSize \leftarrow DataPktSize(\rho)$
    $modify\_parameters()$
  else if *not retransmission* **then**
    compute $\rho(lqi\_map)$
    $MA \leftarrow \rho * sf + (1 - sf) * MA$
    $DataPktSize \leftarrow DataPktSize(MA)$
    $modify\_parameters()$
  end if
end for

In Equation 4, the scaling factor (*sf*) is chosen such that current LQI sample is given more weight than the previous LQI samples and any temporary fluctuation in the current

LQI sample does not immediately affect the packet size estimation. Hence, the scaling factor (*sf*) is given a value of 0.4 based on the experiments. Note, for values less than 0.4 the moving average took longer to converge to an optimal LQI estimate, whereas, for values greater than 0.4 there are increased fluctuations in the LQI estimate. The data message packet size (*DataPktSize*) is determined by the estimated moving average, according to the data mapping in the Table 1. In the case of retransmission, only the nodes with poor LQI requests more packets. So the moving average technique is avoided to reduce any possibility of further retransmissions and the *DataPktSize* is determined according to the current average LQI (ρ) estimate.

$$\delta = \frac{New\ packet\ size}{Current\ packet\ size} \tag{5}$$

After determining the new packet size, the *multiplication factor (δ)* is calculated using Equation 5. If the new packet size is greater than the current packet size, then δ number of current packets are joined together into a single packet and transmitted. If the new packet size is less than the current packet size, then the current packet is split into $\frac{1}{\delta}$ number of packets and transmitted. So in SIMAGE, the current packet can be divided into smaller packets, or two, or more current packets can be combined together into a single packet. To facilitate this, the payload size value for different LQI ranges in Table 1 are chosen such that the bigger packet size values are a multiple of all other smaller packet size values. After determining the new packet size, the transmitter communicates it to the receiver using the data packet header. So, there is an one byte overhead in the data packet in this scheme when compared to the code dissemination protocol in TinyOS, Deluge [5].

On receiving the first data packet of a page, the receiver checks for any change in packet size. If the new packet size value is greater, or less than the current packet size, the receiver computes the δ value and populates the new bit-vector, which consists of the packet identifiers of packets to be transmitted, based upon the packets requested in the old bit-vector and δ value. After updating the new bit-vector it checks whether the packet number of the received packet is requested by this node. If it was requested by this node, it stores the data, otherwise, discards it.

6.6 Secure Dissemination

The three different types of messages involved in the code dissemination process are *advertisement messages*, *request messages* and *data messages*. Securing each of these message types protects against different types of attacks. For instance, securing only the data messages and not providing integrity for the request messages, or advertisement messages enables an attacker to successfully perform a denial-of-service attack by flooding the nodes with request messages, or advertisement messages. In SIMAGE, an increased level of security is ensured by encrypting all three types of messages using RC4 encryption and provide integrity using the CBC-MAC. The CBC-MAC integrity check is fast as it is provided by the CC2420 hardware module. A 64-bit key is used for CBC-MAC, which incurs only 8 bytes of overhead to the overall payload. Whereas, RC4 encryption uses a 128-bit key and being a stream cipher technique, it performs an in-place encryption and does not add any data header overhead.

6.7 Performance Analysis

This section discusses the performance of SIMAGE (with and without the security) under varying link quality conditions and compare it with the existing code dissemination protocols for TinyOS - Deluge [5] and Seluge [6]. Two different experiments were performed to analyze and expose the benefits of SIMAGE:

Experiment 1: An experimental setup consisting of two MICAz sensor motes is used. A binary image of 35,276 bytes is transferred between the nodes using the default code dissemination protocol (Deluge) [5], SIMAGE without security and SIMAGE with security. The three techniques mentioned above are compared with respect to the average number of retransmitted bytes per page and the average transmission time per page. This experiment is performed for different link quality conditions.

Figures 14 and 15 illustrate the results obtained from the first experiment. As seen in Figure 14, for the best LQI range, the default code dissemination protocol [5], SIMAGE with security and SIMAGE without security do not exhibit any retransmissions. Similarly, for the good LQI range, only a small amount of retransmissions are encountered by all three

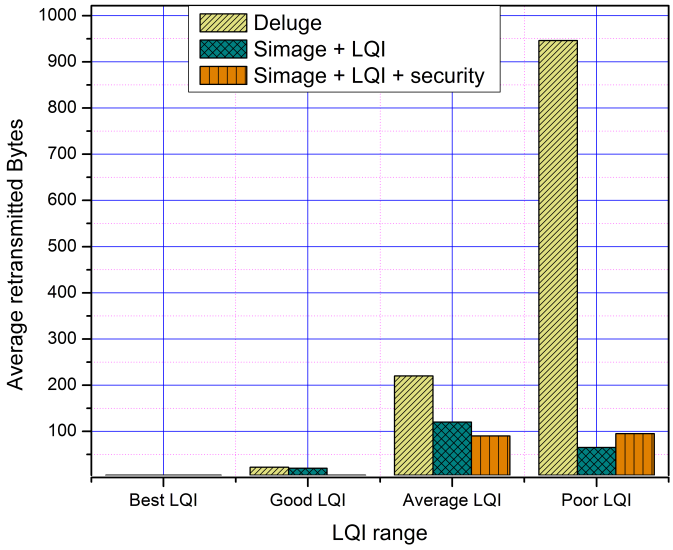

Figure 14: Average retransmitted bytes for various LQI ranges

techniques. As the link quality decreases, Deluge experiences increased retransmissions as compared to SIMAGE with and without security. Furthermore, at very poor link quality conditions, Deluge exhibits a steep increase in the number of bytes retransmitted, whereas SIMAGE with and without security maintains an almost steady number of retransmitted bytes. Moreover, under such low link quality cases, the number of bytes retransmitted using SIMAGE without security is less than Deluge because the packet size is dynamically reduced to a minimal value. It can also be seen that SIMAGE with security has a higher number of retransmitted bytes than SIMAGE without security because of the fact that the former experiences additional security overhead for each packet transmitted.

Figure 15 depicts the same experiment illustrating the variations in the average time taken per page transfer. Under very good link quality conditions, SIMAGE without security proves to provide much faster page transmission than Deluge as larger packets are transmitted with relatively small packet loss rate. SIMAGE with security also proves to be faster than Deluge, however, becomes slower than SIMAGE without security because of the additional security overhead in each packet. As the link quality condition degrades, the time

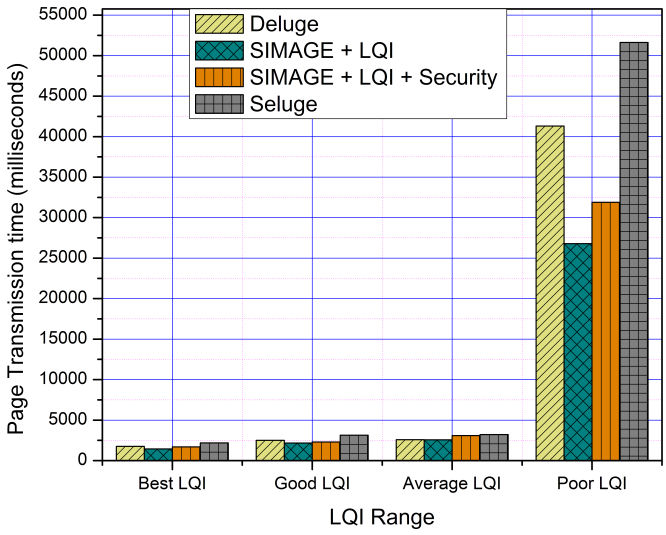

Figure 15: Page transmission time for various LQI ranges

taken per page transfer by all three techniques tends to increase. This can be accounted for by the fact that the number of retransmissions also increase. In the average link quality condition, Deluge and SIMAGE have almost equal page transfer times. This is due to the fact that under average link quality conditions, both protocols transmit packets of almost equal sizes (Deluge has 23 bytes and SIMAGE has 20 bytes). Again because of the security overhead in each packet, SIMAGE with security takes slightly a longer time to transfer a page than the other two schemes. When the link quality further deteriorates and reach the poor link quality condition, Deluge suffers a large number of retransmissions and takes a very long time to transfer a page. Whereas, it can be seen that SIMAGE with and without security transfer a page in a relatively short time. For performance reason, Deluge keeps the radio switched on during code dissemination [5] and any decrease in the code dissemination time reduces the energy consumed by all the nodes in a network. So, reduction in the per page transmission time by SIMAGE, in turn reduces the energy consumption of the nodes and proves that SIMAGE is more energy-efficient than Deluge.

Furthermore, Seluge [6], consumes more time to do integrity check using the SHA-1 algorithm. SHA-1 takes around 15 milliseconds to compute the hash of a packet and the

inter-packet transmission time is increased from 2 to 17 milliseconds to accommodate the computation time [6], which increases the per page transmission time. Using the performance results from Seluge [6], a comparison of per page transmission time is shown in the Figure 15. SIMAGE with security outperforms both Deluge and Seluge under poor link quality conditions.

Experiment 2: The second experiment consisted of three levels of sensor nodes with two MICAz sensor motes at each level. While the previous experiments show the benefits of SIMAGE under various link quality conditions, the primary motive of this experiment was to prove the effectiveness of SIMAGE over the default code dissemination protocol [5] under very good link quality conditions as well. The time taken by the node in the last level to receive the entire binary image (35,276 bytes) is evaluated for both the default code dissemination protocol [5] and SIMAGE.

Table 2: Code dissemination time taken by Deluge and SIMAGE in a multi-hop network

Protocol	**Node**	**Total Image Transfer Time**
Deluge	intermediate	147.492s
	sink	296.379s
SIMAGE	intermediate	96.572s
	sink	198.216s

The results obtained from the second experiment are presented in Table 2. The total time taken for the binary image transfer from source node to intermediate nodes and from source node to sink nodes are evaluated. All the nodes in the experiment are maintained under very good link quality conditions such that the retransmissions are largely reduced. This experiment reiterates that SIMAGE performs better than Deluge in terms of total transmission time for the entire binary image even under good link quality conditions. This also illustrates that SIMAGE makes better use of the channel when the link quality between nodes is good, whereas, Deluge does not make optimum use of the channel under good link quality conditions.

CHAPTER VII

PROVIZ USAGE SCENARIO: BLACK-HOLE ATTACK VISUALIZATION

In this usage scenario, it is discussed how PROVIZ is used to visualize a WSN, and how it enables visual debugging to identify the black-hole attack [27] are shown. To initiate a black-hole attack, first a compromised node advertises a route message to other nodes, claiming to have a shortest route to the cluster head. Then, the compromised node starts receiving the packets from other nodes, whose packets it eventually drops.

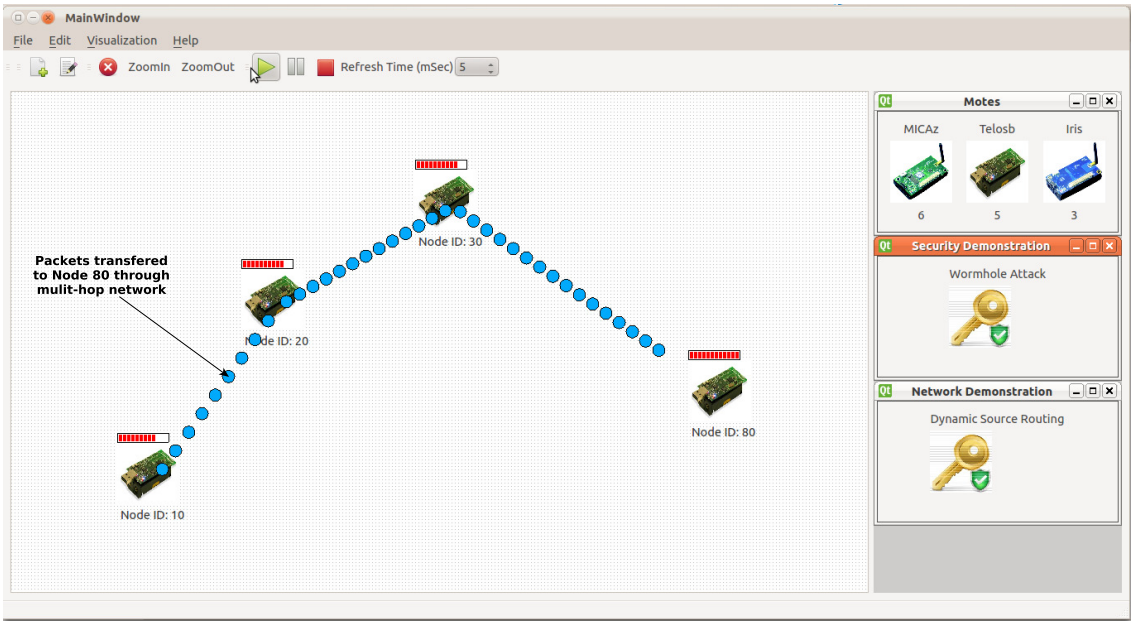

Figure 16: PROVIZ Visualizing a Multi-hop Wireless Transmission

Figure 16 shows the visualization of a WSN with three environment monitoring sensor nodes, with Node IDs 10, 20, and 30 and a cluster head, with Node ID 80. The sensor nodes in the WSN gather the environment information (e.g., temperature, ambient light) periodically and sends it to the cluster head. The sensor nodes use the Dynamic Source

Routing (DSR) [28] protocol, for determining the route to the cluster head and uses source-routing [28] to route the packets to the destination. In DSR, to determine a route to a particular node ID, the sender generates a Route Request (RREQ) message for the destination ID and then broadcasts it. The intermediate nodes, on receiving this message appends their node ID and re-broadcasts the RREQ message, until the destination node receives the RREQ and replies with a Route Reply (RREP) message. The RREP message has the route information that the RREQ message took and it is source-routed to the sender. The sender, on receiving the first RREP message, gets the route information and starts sending the packets.

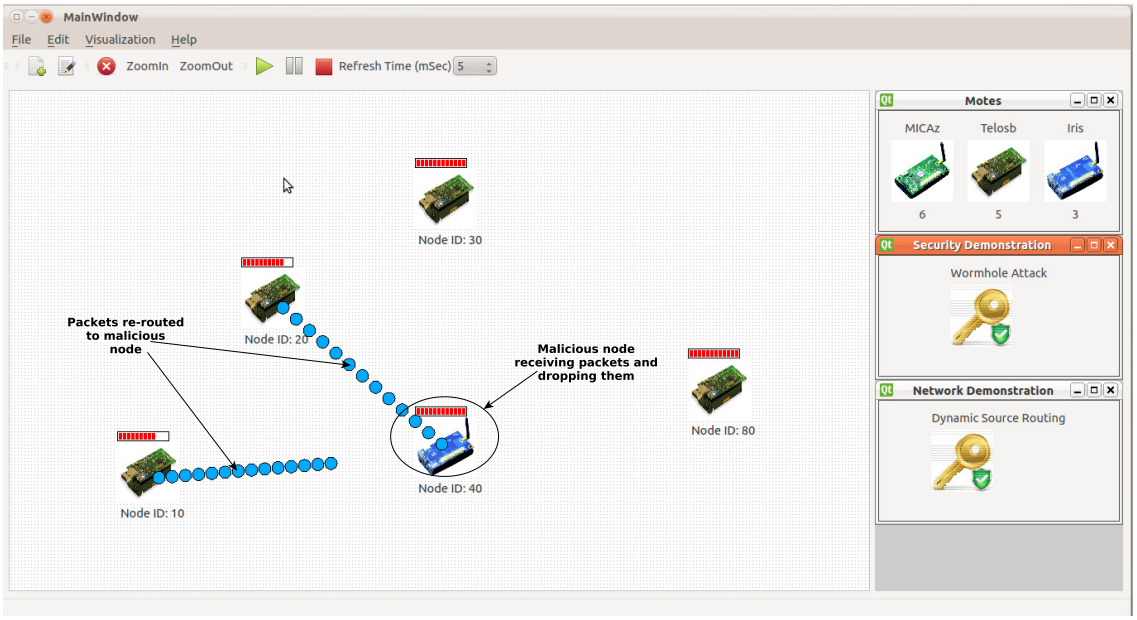

Figure 17: PROVIZ Visualizing a Black-hole Attack

Node 10 in the WSN sends the data to the cluster head, in a multi-hop route through nodes 20 and 30 as shown in Figure 16. In the same network, shown in Figure 17, it is assumed that node 40 is compromised by an attacker to demonstrate the black-hole [27] attack scenario. Although, node 40 is not a single hop neighbor of the cluster head, it sends a route advertisement claiming that it is one hop away from the cluster head. After receiving the route advertisement, node 10 starts sending the data packets through node 40 and the compromised node starts dropping the packets, thus initiating a black-hole attack. When

an attack of this type is launched, it may not be trivial for a researcher to understand this attack and to identify the malicious node. In the mean time, the compromised node might have dropped critical time-sensitive data. The researcher needs to analyze the packet logs continuously to detect an anomaly in the WSN, which is a time consuming process. Also, if the black-hole attack is initiated intermittently, then it will be hard to detect the attack by packet log analysis. However, with the help of PROVIZ visualization tool, the researcher can monitor, or replay the packet transfers from the point when the cluster head stops receiving packets and can visually debug and identify that node 40 is compromised. Figure 17 shows that node 40 receives packets from nodes 10 and 20 and drops the packets, which depicts a black-hole attack. On identifying the black-hole attack, the researcher can use the PROVIZ programming functionality to modify the WSN application and use SIMAGE, provided by the PROVIZ programming functionality, to reprogram the compromised node over-the-air. Thus, the visualization tool can be used as a visual monitoring and debugging tool to detect a network anomaly, and its programming functionality can be used to provide an instant fix, for the attack by reprogramming the sensor nodes.

CHAPTER VIII

CONCLUSION AND FUTURE WORK

8.1 Research Contribution

In this thesis, an integrated visualization and programming framework developed for WSNs, named PROVIZ is introduced. PROVIZ is an open-source framework, which is designed to be modular, scalable, and platform independent. PROVIZ is capable of visualizing a WSN and the packet transfers occurring between the sensor nodes in real time. The PROVIZ visualization tool is generic and extensible, such that it can take packet data input from various sources like live sensor-based sniffers, commercial sniffers (e.g., TI SmartRF packet sniffer [10]), and the OMNeT simulator [9]. PROVIZ can take multiple user-defined packet formats as input and translate the raw packet data from a heterogeneous network into user-readable content. Users can define the packet formats graphically using the PROVIZ Packet Format Specifier. Also, PROVIZ includes built-in extensible visual demo deployment capability that can be shared among the users in the form of XML files. PROVIZ includes an editor window, which can edit a NesC code and MCL [8] script.

Additionally in this thesis, a wireless code dissemination protocol, named SIMAGE is introduced. Our experimental results with existing code dissemination protocol [5] shows that using a fixed packet size will increase the retransmissions between the nodes with poor link quality and hinders the overall code dissemination time. SIMAGE, samples the link quality of the channel around the base station using LQI as a metric and determines the optimal packet size before transmitting a page. The performance analysis of the SIMAGE in the experiments shows that dynamic adaptive packet size reduces the retransmission bytes by 93% and the per page transmission time is reduced by 35% for poor LQI values. Results also show that SIMAGE, during the good link quality condition exploits the channel and disseminates the code faster when compared to the other code dissemination protocols.

SIMAGE with security uses the simple stream cipher encryption algorithm RC4 and a hardware based hashing function CBC-MAC, for providing secure code dissemination. Since, SIMAGE uses a dynamic adaptive packet size technique, the overhead of providing secure dissemination is reduced as compared to Seluge [6]. Our experimental results also shows that SIMAGE with security outperforms the TinyOS code dissemination protocols Deluge [5] and Seluge [6], under the poor LQI ranges.

Finally, a black-hole security attack scenario in a WSN is discussed and analyzed. It is also shown: 1) how PROVIZ could be used for detecting the attack by visual debugging and to aid in providing a software fix; and 2) how PROVIZ framework uses SIMAGE to wirelessly reprogram the compromised node.

8.2 Future Research Direction

PROVIZ will be extended to get live sensor state information like temperature, battery level, etc. An algorithm to implement duplicate packet detection for PROVIZ visualization tool will be implemented. Also, PROVIZ will be improved to include a capability to define notifications to the user whenever a predefined network condition is reached. These features will be helpful when a researcher, developer, or user needs to continuously monitor a WSN to get notifications. For instance, the user can be notified whenever the battery level of a sensor node goes down beyond a certain limit, or when the temperature readings goes beyond certain threshold. Also, the PROVIZ programming functionality will be extended to support graphical programming using drag-and-drop icon-based programming style, for easy and rapid WSN application development. The MCL will be extended to define Simple and Complex Modules for WSN application, and similar to graphical programming tool it will be extended to generate an intermediate XML code segment before generating the NesC code. This feature will enable PROVIZ to generate graphical program templates from MCL code and vice-versa. Also, SIMAGE will be extended to work with sensor nodes running custom WSN applications not created by TinyOS.

REFERENCES

[1] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *International conference on Embedded networked sensor systems*, ACM, 2003.

[2] L. Shu, C. Wu, Y. Zhang, J. Chen, L. Wang, and M. Hauswirth, "Nettopo: beyond simulator and visualizer for wireless sensor networks," *SIGBED Rev.*, vol. 5, Oct. 2008.

[3] L. Ma, L. Wang, L. Shu, J. Zhao, S. Li, Z. Yuan, and N. Ding, "Netviewer: A universal visualization tool for wireless sensor networks," in *IEEE Global Telecommunications Conference*, Dec. 2010.

[4] A. Ruzzelli, R. Jurdak, M. Dragone, A. Barbirato, G. OHare, S. Boivineau, and V. Roy, "Octopus: A dashboard for sensor networks visual control," in *Proceeding of the 14th Annual International Conference on Mobile Computing and Networking*, 2008.

[5] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proceedings of the 2nd international*, pp. 81–94, ACM Press, 2004.

[6] S. Hyun, P. Ning, and A. Liu, "Seluge: Secure and dos-resistant code dissemination in wireless sensor networks," in *Proceedings of the 7th IPSN*, 2008.

[7] "PROVIZ: A Visualization and Programming tool for WSNs." `http://www.ece.gatech.edu/cap/proviz`.

[8] M. Valero, S. Uluagac, V. Subramanian, R. K. Chandrasekar, and R. Beyah, "The monitoring core: A framework for sensor security application development," in *IEEE International Conference on Mobile Ad hoc and Sensor Systems*, Oct. 2012.

[9] "OMNET Simulator for WSN running TinyOS programs." http://www.omnetpp.org/pmwiki/index.php?n=Main.NesCT.

[10] "Ti packet sniffer." `http://www.ti.com/tool/packet-sniffer`.

[11] R. K. Chandrasekar, V. Subramanian, S. Uluagac, and R. Beyah, "SIMAGE: secure and Link-Quality cognizant image distribution for wireless sensor networks," in *IEEE Global Telecommunications Conference*, Dec. 2012.

[12] B. A. Forouzan, *Cryptography and Network Security (1st edition)*. McGraw-Hill, 2007.

[13] "CC2420 datasheet." http://focus.ti.com/lit/ds/symlink/cc2420.pdf.

[14] "Qt: A gui development framework." http://qt-project.org/.

[15] "Qt: A multi-platform tool." http://doc.qt.digia.com/qt/supported-platforms.html.

[16] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," PLDI '03, ACM, 2003.

[17] "Approved draft revision for ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements-part 15.4b: Wireless medium access control (mac) and physical layer (phy) specifications for low rate wireless personal area networks (wpans) (amendment of ieee std 802.15.4-2003)," *IEEE Std P802.15.4/D6*, 2006.

[18] "Network time protocol project." http://www.ntp.org/.

[19] "Infrastructure monitoring wsn." `http://wang.ce.gatech.edu/research.htm`.

[20] "Tinyos documentation." `http://docs.tinyos.net/`.

[21] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler, "Securing the deluge network programming system," in *Proceedings of 5th IPSN*, pp. 326–333, 2006.

[22] M. Passing and F. Dressler, "Experimental performance evaluation of cryptographic algorithms on sensor nodes," in *IEEE MASS*, pp. 882–887, oct. 2006.

[23] R. Venugopalan, P. Ganesan, P. Peddabachagari, A. G. Dean, F. Mueller, and M. L. Sichitiu, "Encryption overhead in embedded systems and sensor network nodes: modeling and analysis.," in *CASES*, pp. 188–197, 2003.

[24] "Ecdsa performance evaluation." `http://discovery.csc.ncsu.edu/`.

[25] D. He, C. Chen, S. Chan, and J. Bu, "Dicode: Dos-resistant and distributed code dissemination in wireless sensor networks," *IEEE Transactions on Wireless Communications*, vol. PP, no. 99, pp. 1–11, 2012.

[26] G. Zheng, D. Han, R. Zheng, C. Schmitz, and X. Yuan, "A link quality inference model for ieee 802.15.4 low-rate wpans," in *IEEE GLOBECOM*, pp. 1–6, dec. 2011.

[27] Y.-C. Hu, A. Perrig, and D. Johnson, "Wormhole attacks in wireless networks," *Selected Areas in Communications, IEEE Journal on*, vol. 24, Feb. 2006.

[28] D. B. Johnson, D. A. Maltz, and J. Broch, "Ad hoc networking," ch. DSR: the dynamic source routing protocol for multihop wireless ad hoc networks, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

VITA

Ramalingam was born on July 8, 1986 in Madurai, India. He received his B.E. in Electrical and Electronics Engineering (EEE) from PSG College of Technology, affiliated to Anna University, Tamilnadu, India, in 2008. He was awarded with the most prestigious *Koorathalwar* award, for his excellent work in academics, sports, and service to the society. After his graduation, he worked with TOSHIBA, India, for three years as a Software Engineer in the Multi-Function Printer (MFP) network solutions group.

He received his M.Sc. in Electrical and Computer Engineering (ECE) from Georgia Institute of Technology, Atlanta, GA, in 2013. During his Masters, he served as a Graduate Research Assistant (GRA) in the Communications Assurance and Performance (CAP) group at Georgia Tech and has several publications in the area of security and Wireless Sensor Networks (WSNs).

In Summer 2012, he did his internship with Qualcomm Innovation Center (QUIC), Bouder, CO. In QUIC, he worked with the Android power management team and after his graduation, he continued working as a full-time Software Engineer at QUIC from July, 2013. Also, he is a trustee of the non-profit organization named Computer Kindness Foundation (CKF), which work towards helping students to get basic education.

PUBLICATIONS

- Chandrasekar, R. K., Subramanian, V., Uluagac, S., and Beyah, R., SIMAGE: Secure and Link-Quality cognizant image distribution for Wireless Sensor Networks, in IEEE Global Telecommunications Conference, Dec. 2012.

- Valero, M., Uluagac, S., Subramanian, V., Chandrasekar, R. K., and Beyah, R., The monitoring core: A framework for sensor security application development, in IEEE International Conference on Mobile Ad hoc and Sensor Systems, Oct. 2012.

- Chandrasekar, R. K., Uluagac, S., and Beyah, R., PROVIZ: An Integrated Visualization and Programming Framework for Wireless Sensor Networks, Submitted at IEEE International Conference on Computer Communication and Networks, Jul. 2013. (under review)